L16: Counting Triangles in MapReduce

scribe(s): *Meysam Taassori*

These days, global pool of data is growing at 2.5 quintillion byte per day and more than 90 percent of this huge pool of data has been produced in the last two years alone [1]. The era of big data has arrived. After [2] explained the file system of Google in this way such that files are split in to various chunks stored in a redundant fashion on a cluster or commodity machines, most of research groups paid attention to big data as a new field of research. Next step was Map Reduce[3] which bound big data with "Map Reduce". After Google introduced Map Reduce in 2004, this new style became synonym with Big Data. Google uses Map Reduce in order to calculate the search indices. In fact, they have the results of search sitting in their clusters and everyday they run Map Reduce to recalculate everything.

16.1 Introduction

Map Reduce is a paradigm or a programming model which is appropriate for processing and generating large data sets [3]. In this model, a "map function" is specified to process a pair of key/value in order to generate a set of intermediate key/value. Then, a "reduce function" merges all these intermediate values which belong to the same intermediate key. This style parallelizes our program automatically and gives us this opportunity to run the programs on a huge cluster of commodity machines. This style also gives an opportunity to programmers to experience parallel programming an its advantages. Another important point about Map Reduce is that its implementation is easy and scalable. For many years, the programmers of Google were busy implementing special-purpose computations that process large amount of raw data. Google has a lot of information including web request logs, crawled documents, and so on. These kind of computations are straightforward but they are very large so they have to be distributed across hundreds of machine to finish in a reasonable amount of time [3].

The researchers in Google found that many of these computations are like map and reduce introduced in Lisp or other functional languages. They came up with this idea that all these computations are working on a record. At first step, all of them must be mapped in to an intermediate key/value pairs; then, a reduce operation is needed to combine all these intermediate records having the same key to generate appropriate information. The most important contribution of this work is to introduce a new model of computation that can easily and automatically parallelize the programs [3].

This essay presents an introduction to Map Reduce then tries to define this model of programming. in session 12-3, we are going to introduce some important usage of this model, the rest of this essay is focusing on one application of Map Reduce called triangle counting.

16.2 Definition

The user of Map Reduce algorithm considers computation as two main functions: Map and Reduce. Programmer gives "map" programming pairs of inputs and this program generates the intermediate pairs. Map Reduce groups all the values related to same key and sends this information to "Reduce" Program. This program which is written by programmer accepts the key and all values related to that key; this program is responsible for merging all this information and converting them to a smaller set of values [3]. Figure 12.1 illustrates these two steps of Map Reduce model.

In other words, Map Reduce has three steps as follows.

Input is a big data set called D which is partitioned into D1, D2, D3, ..., Dm and distributed in different machines in such a way that each machine is dedicated to each portion. Now we can consider three steps of

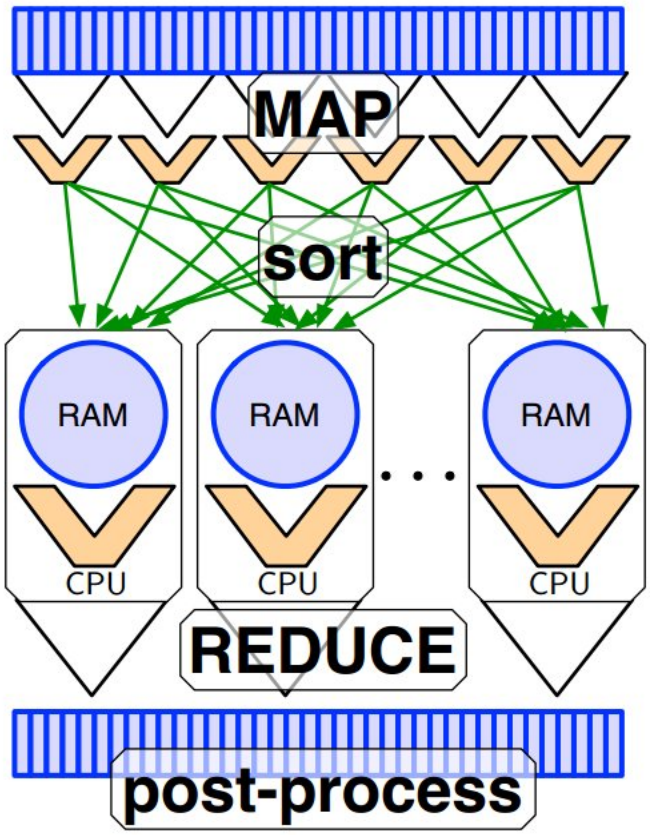

Figure 16.1: MapReduce

Map Reduce in this way

1. Mapping: we assign to $d \in D$ one pair as (Key(d), Value(d)) and we called them k(d) and V(d) respectively. While the mapper is working we can also combine some pairs like (k,v1), (k,v2) which is part of reducing portion but doing it here here improves performance. This step which is subset of mapper called "Combiner".

2. Shuffle: all pairs like (k, v) and (k, V) can be moved to one machine.

3. Reducing: in this step considering (k, v1), (k, v2), ... we can have an output like D(k) = f(v1,v2,v3,...).

As an example for Map Reduce applications, lets consider we are going to count the number of a words in a large document. In this case, our pairs contain one key as a word we are looking for and value which shows the number of that word in the document. So we partition that large document which is our input into different smaller parts and these smaller parts are mapped into different machines. In mapping step, each machine finds the number of that word as a pair (Key, Value). In reduce stage, we easily count the number of values in pairs having same key. This result shows the number of the word called key in that huge document.

16.3 Application of Map Reduce model

There are many applications using Map Reduce to run more efficiently while enjoying from using several parallel machine simultaneously. Distributed Grep is one of these applications which is pretty similar to

word counting. In another example, Map Reduce is able to generate large-scale PDF files quickly an easily. When New York Times needed to generate PDF files of all its archive in the format of scanned image, Map Reduce algorithm was appropriate for this purpose. Map Reduce also can be helpful for artificial intelligence applications when we want to compute some statistics. Handling the large sets of data including many roads, intersections is one of the most significant problem of Google Map. Finding the roads which is connected to one given intersection, rendering the map tiles, and finding nearest say restaurant to a given address can be done by Map Reduce faster and more efficiently. In this case, our big data contains a gigantic list of roads and intersections. Mapper produce some pairs containing road and intersection or road and road. Then we can sort them by key. In reduce step, a list of pairs with the same key can be produced which we can derive the useful output from.

One of the most important usage of Map Reduce is "Page Ranking". This algorithm is a way of helping Google estimate and measure the importance of one web page. This algorithm can estimate how one page is important by counting the number of links to that page and considering the quality of those links. Although it is not the only algorithm, it is the first and the most well-known algorithm to assign ranks to pages. In this algorithm, we consider entire Internet as a big matrix M(n*n) where each row is dedicated to one page and the columns show the page linking that web page. The page rank of one web page is shown by q which is vector (n*1). we are going to calculate q by using Markov chain and because the size of memory is not enough to save all M and q we are going to apply Map Reduce algorithm.

Another important application of Map Reduce is triangle counting which we are going to explain completely. The rest of this essay is trying to explain this usage of Map Reduce.

16.3.1 Triangle Counting

One important factor of each social network is "Clustering Coefficient" which shows that how much community around one node is crowded; this calculation can be tuned to counting the number of triangles around one particular node in the graph[4]. Since this graph is too big to fit into the memory, and we want to run it in parallel, we are going to use Map Reduce. [4] tries to fit this problem into Map Reduce Modeling to enjoy parallel programming. In this sub chapter, we are going to introduce precisely the problem of triangle counting; then, we will show how it is possible to fit this problem into Map Reduce Model. Finally, we will talk about the experimental evaluation of this algorithm.

Calculation of clustering coefficient

The clustering coefficient indicates the degree to which a node's neighbors are themselves neighbors. This factor can effect the entire social network. For example, in a tightly-knit community, if one member does something in an offending manner, it is more effective in social network because more members know him and would be informed of this behavior which is against the society. Measuring this factor can be converted to a simple problem which is triangle counting.

To calculate this factor of graph, we consider G= (V, E) as an unweighted, undirected simple graph and let n = |V |, m = |E| and Γ(v) is the set of neighbors of v meaning that Γ(v) = { w $\in$ V | (v,w) $\in$ E } and dv = $| \Gamma(v) |$. In this case, cluster coefficient (cc(v)) for a node v $\in$ V can be defined by

$$\binom{dv}{2} cc(v) = | \{(u, w) \in E \mid u \in \Gamma(v), w \in \Gamma(v)\} | \tag{16.1}$$

It is evident that a pair like (u, w) contributes in the clustering coefficient of node v, first of all, they should be connected to v and secondly they must be connected to each other; therefore, we can infer that these nodes u,w,v can form a triangle. As a conclusion, three nodes u, v, w can shape a triangle if $(u, v), (u, w), (v, w) \in E$. all methods proposed in [4] are based on this conclusion.

16.3.2 Sequential Algorithms

At first, [4] explained some sequential solutions for triangle counting and then it uses Map Reduce to make these solutions parallel.

NodeIterator

This algorithm exactly works based on conclusion mentioned above. In this algorithm, we look for neighbors of node v which are connected to each other. We need to pay attention to this point that in this algorithm each triangle counts 6 times. Analysis of the running time in this algorithm shows that this algorithm can run in $O(\sum_{v \in V} d_v^2)$. Evidently, this algorithm, is not practical for real-world massive graphs; this algorithm is shown as follows

Algorithm 16.3.1 NodeIterator(V, E)

```
T = 0 ;
for v ∈ V do
  for u ∈ Γ(v) do
    for w ∈ Γ(v) do
      if ((u, w) ∈ E) then
        T = T + 1/2;
return(T/3);
```

NodeItrator++

The most important weakness of this algorithm is its running time which makes it impossible for some real gigantic graphs. some other attempts have been done to improve this algorithm; for instance, [5] proposed an algorithm to improve this weakness. To improve the baseline algorithm, we need to note that in that algorithm, each triangle is counted 6 times; to prevent form these unnecessary counts, [5] proposed an algorithm that the lowest degree node just is responsible for making sure that triangle is counted. this algorithm is shown as follows

Algorithm 16.3.2 NodeIterator(V, E)

```
T = 0 ;
for v ∈ V do
  for (u ∈ Γ(v)) and (u > v) do
    for (w ∈ Γ(v)) and (w > u) do
      if ((u, w) ∈ E) then
        T = T + 1/2;
return(T/3);
```

[5] proved that the running time for this algorithm is $O(m^{\frac{3}{2}})$.

16.3.3 Map Reduce Algorithms

All algorithms mentioned in previous session have an assumption that the graph is small enough to be saved totally in the memory if a machine. For massive graphs, this assumption does not work anymore, so researchers are looking for some new algorithms appropriate for parallel computing to run in several machines. Map Reduce model is an appropriate algorithm to remedy this problem. In this subsection, we are going to explain NodeIterator++ algorithm in Map Reduce model and another algorithm applying Map Reduce to run triangle counting for huge graphs.

NodeIterator++ in Map Reduce

To implement NodeIterator++ by using Map Reduce mode, we need to define two rounds as follows:

- Round 1: Generate the possible length two path based on every node in graph in parallel.
- Round 2: check which paths found in first round has potential of being as a triangle. That is, we are looking for which paths of round 1 can be closed by another edge to generate a triangle.

For this purpose, we need to do Mapping and Reducing twice. In first Mapping, we are looking for neighbors of a node having less degree rather than that node. The output of first mapping step is some pairs like ¡u,v¿ which shows that u is neighbor of v with less degree. Then, in first reduce step, we need to find some pairs containing two nodes both of which are connected to v. Therefore, in this step we can find all length two paths.

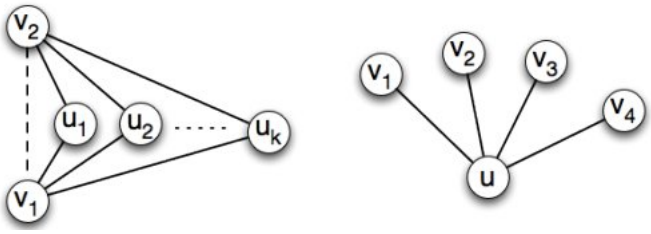

Figure 16.2: Map and Reduce of first round

In another words, first mapper sends all neighbors of each node to the first reducer. As shown in figure 12-2 (right side), all neighbors of u, v_1, v_2 and so on are known in this step. then, the first reducer generate all length two paths in the form of $\prec v1, v2 : u \succ$
The output of first reduce step is delivered to second mapping stage. The second mapping is responsible for making the output from two types of information to second reducer; first, the pairs in the form of$\prec v1, v2; u \succ$ and second the information of edges in format of $\prec v1, v2; \$ \succ$. The second mapper generates output in the format of $\prec v_1, v_2; u_1, u_2, ..., u_k, \$ \succ$. Figure 12.2 (left side) shows the graph of this output. The reducer of second round is in charge of checking which three nodes are able to generate the triangle. This reducer has enough knowledge to find the nodes which are able to generate triangle. Pseudo code of this algorithm is shown as follows

Partition Algorithm

Partition Algorithm is another algorithm which uses Map Reduce to count the triangles in a massive graph but the most important privilege of this method is that it uses Map Reduce algorithm just once rather than the previous algorithm which uses Map Reduce algorithm twice let assume that the massive graph has n nodes and we can partition them in to b distinct subsets named $V_1, V_2, V_3, ..., V_b$ in such a way that $V = V_1 \cup V_2 \cup V_3 \cup V_4 \cup ... \cup V_p$ where if $i \neq jthenV_i \cap V_j$. We also denote $V_ijk = V_i \cup V_j \cup V_k$. Let $E_ijk = \{(u, w) \in E : u, w \in V_ijk\}$ be the set of edge between nodes $V_i, V_j, andV_k$ and let defined a graph $G_ijk = (V_i \cup V_j \cup V_k, E_ijk)$ on V_ijk.

In this algorithm, for each triple of integers $1 \preceq i \preceq j \preceq k \preceq n$ mapper sends all edges whose both nodes are in $V_i \cup V_j \cup V_k$ to one reducer called R_ijk. Evidently, each producer has a smaller graph to find the triangles. each producer is in charge of counting the triangles in small graph emitted to it.

As shown in figure 12.3, some edges might be in more than one partition and in this case more than one reducer might receive that edge; hence we need to count the weighted triangles which [4] proofs that after weighting, each triangle can be counted just one time and no more. to proof that, let consider triangle

Algorithm 16.3.3 MR_NodeIterator++(V, E)

Map 1: Input: $\prec (u, v); 0 \succ$
if $(v \succ u)$ **then**
  emit (u;v)
Reduce 1: Input $(v; S \subseteq \Gamma(v))$
for $(u, w) : u, w \in S$ **do**
  for $(u, w) : u, w \in S$ **do**
    emit (v; (u, w));
Map 2:
if (Input of type (v; (u, w))) **then**
  emit ((u, w);v)
if (Input of type ((u, v);0)) **then**
  emit ((u, v), $);
Reduce 2: Input $((u, w); S \subseteq V \cup \{\$\})$
if $(\$ \in S)$ **then**
  for $v \in S \cap V$ **do**
    emit (v; 1);

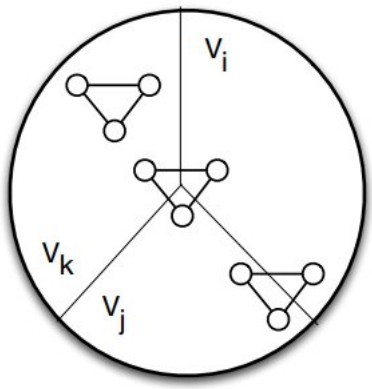

Figure 16.3: Partition Algorithm

Algorithm 16.3.4 MR_GraphPartition(V, E, n)

Map 1: Input: ((u, v); 1)
Let h(0) be a universal hash function into [0,1]
$i \leftarrow h(u)$;
$j \leftarrow h(v)$;
for $a \in [0, 1]$ **do**
  for $b \in [a + 1, 1]$ **do**
    for $c \in [b + 1, 1]$ **do**
      if $\{i, j\} \subseteq \{a, b, c\}$ **then**
        emit ((a, b, c); (u, v))
Reduce 1: Input $((i, j, k); E_ijk \subseteq E)$
Count triangles on G_ijk
for `every triangle` (u, v, w) **do**
  $z \leftarrow 1$
  if $(h(u) \doteq h(v) \doteq h(w))$ **then**
    $z \leftarrow \binom{h(x)}{2} + h(x)(n - h(x) - 1) + \binom{(n-h(x)-1)}{2)}$
  else if ($h(u) = h(v)$ or $h(v) = h(w)$ or $h(u) = h(w)$) **then**
    $z \leftarrow n - 2$
  Scale triangle (u, v, w) weight by $1/z$

(w, x, y) whose nodes are located in $V_h(w), V_h(x)$ and $V_h(y)$. It worth mentioning that $h(x)$ is a hash function to map each node to the portion where it belongs to. If the nodes are located in completely different zones, i.e., $h(w) \neq h(x), h(w) \neq h(y), h(x) \neq h(y)$ then each triangle would be consider just one time. If the nodes are mapped in two zone,i.e, $h(w) \neq h(y), h(x) = h(y)$, this kind of triangle would be counted $n - 2$ times; finally, if $h(w) = h(x) = h(y)$ this triangle would be counted $\binom{h(x)}{2} + h(x)(n - h(x) - 1) + \binom{n-h(x)-1}{2}$ times. Therefore, if we scaled the number of triangles by $1/z$ where z is one of these numbers mentioned above depending on the number of different zones nodes belongs to, each triangle would be counted exactly one time. The pseudo code of this algorithm is mentioned as follows

Bibliography

[1] IBM. IBM "What is big data? bringing big data to enterprise" . http://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/

[2] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung "The Google file system" . Proceedings of the nineteenth ACM symposium on Operating systems principles, page 29–43. New York, NY, USA, ACM, (2003)

[3] Jeffry Dean and Sanjay Ghemawat,"Map Reduce: Simplified Data Processing on Large Clusters" *V*olume 51 Issue 1, January 2008, Pages 107-113 ACM New York, NY, USA

[4] Siddan Suri, Sergei Vassilvitskii, "Counting Triangles and curse of the last Reducer", Proceedings of the 20th international conference on World wide web Pages 607-614 ACM New York, NY, USA WWW 2011

[5] Thomas Schank. Algorithmic Aspects of Triangle-Based Network Analysis. PhD thesis, University at Karlsruhe (TH), 2007.